Geometry-Based Watermarking of 3D Models

Oliver Benedens
Fraunhofer *Institute for* Computer Graphics

This article addresses the fundamentals of geometry-based watermarking. It presents a watermarking algorithm that modifies normal distribution to invisibly store information in the model's geometry.

Current watermarking technology focuses on media types like still **images,**[1-4] and **video**[5] and audio streams? Obviously, more and more CAD-based 3D data is entering the World Wide Web, mostly as **Virtual** Reality Modeling Language **(VRML)** scenes. Accordingly, companies or copyright owners who present or sell their products in virtual space will face copyright-related problems. The straightforward demand is to **prevent** their SD-based (catalogue) material from unauthorized **(unlicensed)** use. Illegal use of copyrighted material can be prevented by introducing a licensee's identity into data. The copyright owner can then identify the source from which illegal copies spread.

VRML scenes involve at least three different media types that are candidates for watermark insertion: audio samples, textures and **background** images, and 3D geometry (model) based data. Data of the first two types might be removed from a VRML scene without significant loss of the scene's value. Attackers could easily exchange textures and audio files suspected of containing watermarks for new (unwatermarked) ones. Because the geometry-based components usually take most of the effort in generating a VRML scene, those components make up most of its value.

Private and public watermarks

In general, watermarks divide into private and public watermarks.

Private watermarks

A private (secret) watermark may contain information for identifying the licensee or to prove ownership in disputes. Retrieval of secret watermark information requires at least one secret key, known only to the embedder. A private watermark puts heavy demands on a watermarking algorithm regarding robustness, **although** the demands for **capacity** are relaxed. Embedded information usually includes licensee-identifying serial numbers or hash values. In general, a serial number is just a pointer or link to externally stored information, such as a customer record.

Public watermarks

A public watermark is retrieved by the receiver (licensee) of copyrighted material. It usually contains copyright or licensing information, such as the identifier of the copyright holder, the creator of the material, or a link (URL) through which to fetch more related information. It may contain a serial number that uniquely identifies material to registration entities. Retrieving a public watermark requires no information but the model data itself plus a specific key, unique among material generated by one or various creators or copyright holders.

In respect to 3D model-based data, a public watermark replaces headers or sections in object file formats that might contain arbitrary user-defined information, such as comments. This information is likely to be stripped off during format conversions.

A public watermark puts heavy demands on a **watermarking** algorithm regarding capacity. Because a public watermark provides additional copyright-related information for receivers and doesn't aim to prove ownership or identify licensees, the requirements regarding robustness are relaxed.

In comparison to audio, video, and still-image data, the task of watermarking 3D models faces some specific problems:

- You must deal with a low volume of data (with respect to separate objects).
- Handling and editing may involve a variety of complex geometrical or topological operations. A variety of tools are available for this purpose-modelers.
- No unique representation of model data exists. Meshes consisting of different vertices, edges, and faces can represent a model surface, and vertices, for example, can be moved randomly by relatively large amounts without degrading overall visual quality.
- No implicit ordering of data exists. Audio and video data are sequenced in time series, while data in still

Form SF298 Citation Data

Report Date *("DD MON YYYY")* 01011999	**Report Type** N/A	**Dates Covered (from... to)** *("DD MON YYYY")*

Title and Subtitle Geometry-Based Watermarking of 3D Models	**Contract or Grant Number**
	Program Element Number
Authors	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Fraunhofer Institute for Computer Graphics	**Performing Organization Number(s)**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Monitoring Agency Acronym**
	Monitoring Agency Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Document Classification unclassified	**Classification of SF298** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** unlimited
Number of Pages 11	

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1/1/99	Article

4. TITLE AND SUBTITLE
Geometry-Based Watermarking of 3D Models

5. FUNDING NUMBERS

6. AUTHOR(S)
Oliver Benedens

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

IATAC
Information Assurance Technology Analysis Center
3190 Fairview Park Drive
Falls Church VA 22042

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Technical Information Center
DTIC-IA
8725 John J. Kingman Rd, Suite 944
Ft. Belvoir, VA 22060

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE
A

13. ABSTRACT ***(Maximum 200 Words)***

This article addresses the fundamentals of geometry-based watermarking. It presents a watermarking algorithm that modifies normal distribution to invisibly store information in the models geometry.

14. SUBJECT TERMS
Watermarking,

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	UNCLASSIFIED	UNCLASSIFIED	None

images and video frames (pixels) are scanline-ordered. Model data like vertices, edges, and faces can be ordered, but this requires at least a fixed orientation and eventually a position of data in space.

Some problems closely resemble those of other media types:

- The process of lossy compression is likely to modify 3D model data. To achieve adequate rendering speeds, the volume of data is reduced by applying polygon simplification. This resembles still images undergoing wavelet or JPEG (Joint Photographic Expert Group) compression, and video and audio streams undergoing MPEG (Moving Pictures Expert Group) compression.
- Synchronization problems occur. During the embedding process performed by a watermarking system, the algorithm might detect locations unsuitable for information embedding. The algorithm might either try to embed some sort of "not used" watermark or skip these locations. For 3D model data it seems a hard problem to define a reliable criterion that tells you which of a given sequence of embedding locations have been used. However, such a criterion is crucial for realization of public watermark systems. Without it an algorithm must rely completely on error correcting codes and/or redundant embedding.

Required properties of 3D watermarking systems

A watermarking system designed for 3D model data should have the following properties at a minimum.

Capacity

The system should allow embedding of nontrivial amounts of data. Embedding serial numbers that identify buyer or licenser identities requires at least 32 bits of capacity. Proving ownership requires enough capacity to store a hash value (for example, 128 bits for the MDS-Message Digest-hash function and160 bits for the SHA-Secure Hash Algorithm). Space requirements of information embedded in public watermarks range from a global model registration number of 32 bits up to a URL from which to get copyright- or licensing-related information, which could take 256 bits or more.

A class of watermarking systems, called statistical approaches, hashes information of arbitrary length and feeds some sort of random number generator with it. This generator yields locations where data is modified in respect to some global statistical measures like mean value and variance. These systems permit testing for the presence of watermarks whose contents are known in advance. While these systems might claim to have unlimited capacity, they have drawbacks. For example, identifying the licensee of a model found on the net requires testing all licensers' identities (generator seeds).

Robustness

As a matter of principle private watermarks should exhibit resistance to all geometric or topological operations that don't substantially degrade the model's visual quality. (Most of these operations have already appeared elsewhere.[7]) The list includes

- Rotation, translation, and uniform scaling
- Polygon simplification (often needed to achieve adequate rendering speed)
- Randomization of points
- Re-meshing (re-triangulation); generating equal shaped patches with equal angles and surface
- Mesh smoothing operations
- Cut (sectioning) operations-removing parts of the model as in backface culling
- Local deformations

Other, more complex geometrical operations will likely degrade visual quality and usability. Thus a watermarking system's primary goal is definitely not robustness with respect to the following operations:

- Shearing
- Nonuniform scaling along arbitrary axes
- Projections, as projection on a plane
- Global deformations

Recommended properties of 3D watermarking systems

In addition to the required properties, 3D watermarking systems would benefit from further capabilities, as follows.

Background processing and suitable speed

Embedding and retrieving watermarks should be possible without user interaction. Automated search of Web sites and databases for watermarks performed by robots ("agents") are an important application in monitoring the use of both legal and illegal copies and in enforcing the copyright. The ultimate goal in this respect is real-time monitoring. However, this puts heavy demands on watermarking systems' execution speed and storage requirements.

Embedding multiple watermarks

Real-world applications might demand the possibility of embedding multiple watermarks. This occurs in the producer/resellers pipeline, in which producers embed a secret watermark identifying resellers and resellers embed one identifying customers or end licensees. Ideally; the watermarks should not interfere with each other (with high probability) even if producers keep the details of their embedding parameters secret.

Minimum knowledge of a priori data

An ideal system requires only model data and knowledge of a key for watermark retrieval. The key may be specific to the model's creator or company, model class, model itself, and/or licensee. All necessary parameters, such as seed values for generators, derive from this key.

In a public watermark system all of a creator's models may share a unique key. Alternatively, the system might use a key common to models from various creators.

Unfortunately, a retrieval system may demand knowledge of more a priori data:

- Knowledge of the model itself, especially model-specific embedding locations to avoid synchronization problems
- Original model data or at least parts of it (the feature vector) for reorientation, rescaling, and comparing features of a watermarked copy with the original

Such a retrival system has reduced monitoring and background processing features. Decentralized retrieving of watermarks gets complicated, especially when it involves accessing large amounts of original model data organized in databases.

Minimum preprocessing overhead

An ideal system should allow immediate access to embedded watermarks, without preprocessing model data. Such preprocessing might involve transforming the model data representation, recognizing the model, correcting surface normals, and reorienting and scaling with respect to the original.

Status quo of 3D geometry-oriented watermarking systems

The previous two sections proposed required and recommended properties of systems watermarking 3D geometry. Ohbuchi, Masuda, and Aono[7] produced the first publication on 3D watermarking, in which they proposed a large variety of techniques. The techniques described fall roughly into mesh altering, topology altering, and visible pattern embedding methods.

The mesh-altering methods, especially their proposed Tetrahedral Volume Ratio (TVR) algorithm, show near-optimal properties with respect to capacity, execution speed, and monitoring capabilities. The significant drawbacks include vulnerability to re-meshing operations, polygon simplification, and point randomization. Nevertheless, their algorithm proves well suited for embedding public watermarks.

Topology altering methods, which basically embed information by cutting holes in a mesh, are trivially vulnerable against attacks that scan for these holes. The visible pattern embedding method, Mesh Density Pattern[7] (MDP), is vulnerable to a re-meshing attack that generates patterns with mostly identical shapes (angles and size).

Embedding private watermarks by altering normal distribution

The watermarking system I propose in this article handles embedding private watermarks. With this system I wanted primarily to achieve robustness against

- Randomization of points
- Mesh altering (re-meshing) operations or attacks
- Polygon simplification

The central idea driving this system is the observation that the operations mentioned may cause large changes in model vertex and face set configuration, adjacencies, and eventually topology.

A 3D object can be seen as a collection of surfaces with arbitrary size and orientation. There exists nearly an infinite amount of meshes representing or approximating one particular set of surfaces, with varying perceived quality.

The idea is to use collections of surfaces as an embedding primitive. If a model's representation changes in response to the operations mentioned, the new vertex face set configuration has to maintain global surface characteristics regarding size, orientation, and curvature, at least of perceivable features, otherwise a loss of visual quality will occur. (Of course, visual quality depends on viewing distance. Simply assume that a model subjected to extensive alterations will be used for the same purpose, under the same conditions, as the original.)

The system proposed in this article introduces modifications in object-model normals distribution to achieve independence from one particular mesh representation. It requires a mesh representation of a 3D model consisting entirely of triangle patches as input. This original model is denoted by M. A watermark, W-a bit string of arbitrary contents and length-is embedded in this mesh by performing certain displacements of points, which in turn introduce specific changes in mesh surface normals distribution. The resulting mesh M' again consists of triangle patches, with no topological or adjacency (vertices or faces) changes with respect to M.

Before describing the embedding process, I need to define and explain the terms "bin" and "extended Gaussian image" (EGI). As already mentioned, the system uses a collection of surfaces as an embedding primitive. These collections are generated by grouping model normals into distinct sets called bins. A bin is the entity for embedding one bit of watermark data. It's defined by a center normal in 3D space normal and possibly a radius. The decision on which set to assign a normal is ruled out by the difference (in angles) of the normal to each set's center normal, called bin center. A normal gets assigned to the bin with the minimum difference.

The bins can be constructed in a general way by tessellating the unit sphere, for example by projecting subdivided regular polyhedrons (platonic solids). This results in bins with equal area size and angles. While these properties prove useful in determining object pose and orientation, they aren't necessary for embedding watermarks.

Instead, a random number generator may generate a sequence of bin center normals. Or, we might search spherical data for suitable centers, which fulfill restrictions regarding minimum surface curvature, the number of normals within a certain radius around the center, and a suitable normal vector distribution-bin data that don't contradict the watermark embedding or retrieving procedure.

The discrete approximation of the EGI,[8,9] called an *orientation histogram,* provides a graphical representation of the described sampling of model normals. Each vector corresponds to a bin center, its direction corresponds to the bin center normal, and its length corresponds to the sum of face sizes of normals contained in the bin. Figure 1 shows an example for this representation.

Although not correct in terms of definition, for convenience I'll use the term EGI as a synonym for the orientation dianram in the following sections.

The embedding and retrieval processes consist of several stages, as follows:

General outline of the embedding procedure:
(El) Calculate consistent surface patch normals
(E2) Sample model normals to bins
(E3) Apply core watermark embedding algorithm

General outline of retrieval procedure:
(R1) Calculate consistent surface patch normals
(R2) Transform model into spherical representation (EGI) and adjust model orientation
(R3) Sample model normals to bins
(R4) Apply core watermark retrieval algorithm

To explain the embedding and retrieval processes, I'll go through the procedures according to these steps.

Calculating consistent surface normals (E 1)

The method depends on consistent surface patch normal directions. Modelers (or attackers) may produce models with patch normals not pointing in an outward direction; in the extreme, normals point randomly in- or outwards. We don't need correctly oriented normals (all pointing outwards) so much as we need reliable directions. That means the same model always yields the same normal directions for its patches.

To accomplish this, first calculate the center of mass (of the point set) for a given model M. For each patch cast a ray from the center of mass through the center of the patch and determine angles to both possible normals. Choose the normal with the greater angle as the surface patch normal. The triangle points are ordered counterclockwise.

For convex objects, the normals determined in this way will all point outwards.

Sample model normals to bins (E2)

Conventions: In the following we always operate on normalized vectors.

Define N_B as the total number of bins, $BC_i \in \mathbb{R}^3$ as bin centers, and $R_i \in [0, \ldots, \pi/2]$ as radius (an angle measured in radians; $i = 1, \ldots, N_B$). Each bin i ($i = 1, \ldots, N_B$) is assigned all the model's normals whose angle difference to the center normal is less than R_i. Let BN_i be the total number of normals sampled to bin i. Let $BP_{ij} \in \mathbb{R}^3$ ($j = 1, \ldots, BN_i$) be the normals sampled in bin i.

Assume bins do not overlap. This can be simply assured for a given set of bin centers by assigning each bin center a radius less than the minimum angle difference to all other bins. Implicitly, surface normals will exist that do not fall in any bin.

Core watermark em bedding algorithm (E3)

For embedding one bit of information, the mean normal of a bin-the center of mass-is moved in a certain direction. Embedding a bit string of length n requires center of mass modifications in n bins. These modifications are performed by displacing vertices in the mesh, thus altering adjacent face (triangle) normals, which in turn cause changes of the center of mass in their assigned bins.

1 The left image shows an orientation histogram of Wagner's bust as shown in the right image. The bins were constructed by projecting an icosahedron on a unit sphere and further subdividing triangles two times, yielding 320 bin centers. The large normal vector pointing downwards results from the large triangles forming the base of the bust.

The contents of a bin can be transformed from 3D into a 2D representation with the bin center as origin and bin boundary the unit circle of radius 1. Since the 2D representation has advantages for illustrating the embedding and retrieval algorithm, the next section describes the transformation, which I also used in further illustrations.

Transforming bin normals from a sphere surface representation onto a circle in $\mathbb{R}^2$. The transformation for the bin with index i ($i = 1, \ldots, N_B$) follows.

First calculate two vectors, $X_1, X_2 \in \mathbb{R}^3$ orthogonal to each other and bin center BC_i. These vectors become the two principal axes of the 2D coordinate system.

Rotate BC_i onto they axis by performing two successive rotations around the z and x axes. Apply the inverse of this transformation to thex and z axes, yielding the two principal axes, X_1 and X_2, of the 2D coordinate system.

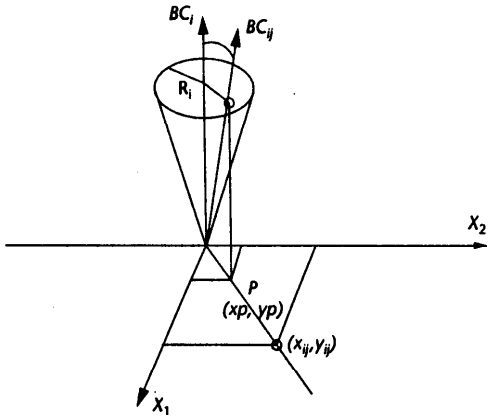
Next transform the bin contents, normalized 3D vectors BP_{ij} ($i = 1, \ldots, N_B, j = 1, \ldots, BN_i$) into its 2D coordinate pair $p_{ij} = (x_{ij}, y_{ij})$ by

$$h = BC_i * BP_{ij}, l_1 = \cos^{-1}(h), P = BP_{ij} - h * BC_i$$

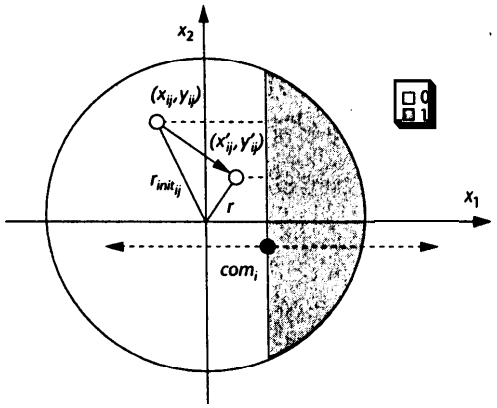
We can suppose that h lies between 0 and 1 (differences of bin center and sampled normals are supposed to be less than 90 degrees).

$$\mathrm{Xp} = X_1 * BP_{ij}$$
$$yp = X_2 * BP_{ij}$$
$$l_2 = |(xp\ \ yp)|$$
$$x_{ij} = \begin{cases} xp * \dfrac{l_1}{l_2} & l_2 > \varepsilon \\ 0 & l_2 < \varepsilon \end{cases}$$
$$\mathrm{yij} = \begin{cases} yp * \dfrac{l_1}{l_2} & l_2 > \varepsilon \\ 0 & l_2 < \varepsilon \end{cases}$$

with the appropriate value for ε (for example, 10").



2 Transformation of bin contents from 3D space onto the plane.



3 Embedding one bit of information in a bin by pushing the center of mass in a certain direction. You've successfully coded a "1" bit if the center of mass moves into the marked section of the circle.

This transformation appears in Figure 2.

Let $S=s_1, \ldots, s_{N_B}$, $s_i \in \{0,1\}$, i= 1, . . ., N_B be the bit-string to be embedded in model M.

Next for each bin i calculate the center of mass $com_i = (cx_i, cy_i)$:

$$com_i = \frac{1}{BN_i} \sum_{j=1}^{BN_i} p_{ij}, (i=1,\ldots,N_B)$$

The distance of each point to bin center is

$$r_{init_{ij}} = \sqrt{x_{ij}^2 + y_{ij}^2}$$

Next, an optimization process iterates over the model's set of vertices, as described in the next section.

Detailed description of the embedding process. The embedding process seeks to move the center of mass of each bin (in the 2D representation described above) to the left (bit 0) or right (bit 1) side of the initial position. It does this by displacing vertices, which in turn modify surface normals. Figure 3 illustrates this process.

At the core of the optimization process, applying an implementation of the multidimensional downhill **simplex**[10] lets you explore a given vertex's neighborhood for a local minimum of costs.

The detailed embedding process is presented by describing three functions as pseudocode: **main()**, optimizeVertex(), costFunc(), and **costs()**. A fifth, multidimensionalDownhillSimplex()-an **implementation** of multidimensional downhill simplex-is called by the function optimizeVertex() to search a vertex's surroundings for a local cost minimum and itself calls costFunc() to evaluate the costs of certain vertex displacements.

The (calling) dependencies of these functions follows:

```
main ()
    optimizeVertex ()
        costs ()
        multidimensionDownhillSimlex ()
            costFunc ()
                costs ()
```

During the optimization process only the location of vertices in mesh M changes. The topology remains unaltered, with adjacencies unchanged. No degeneracies are introduced.

The process also maintains sets containing the original and normal values of triangles as well as their 2D counterparts.

main():
δ = initial search range
for i = 1; i < number of iterations; i++
  for each point P in model
    forj = 1; j < number of refinements; j++
      optimizeVertex(P, δ)
  reduce δ

The multidimensionalDownhillSimplex() function evaluates δ. It determines the search range radius and therefore restricts search space. After each iteration through all vertices of the mesh, this value decreases. The desired effect resembles optimization using simulated annealing methods.

This function iterates a number of times through the model's point set. The function **optimizeVertex()** is called with the current point and δ as parameters. **optimizeVertex()** tries to find a new local minimum with respect to the cost function **costFunc()** in the neighborhood of P. If it succeeds, **optimizeVertex()** updates P with new point coordinates for which a new minimum of costs was reached (it alters the mesh). **optimizeVertex()** iterates this process a number of times with the newly found minimum as the starting point.

optimizeVertex (Vertex P, Factor 6):
initialCosts = costs(P)
P' = multidimensionalDownhillSimplex (initialCosts, P, δ)
alter mesh by exchanging P with P'

This function is just a wrapper for **multidimensionalDownhillSimplex()**. It calculates initial costs caused by the initial position of point P and calls **multidimensionalDownhillSimplex()** to find a new local minimum. The latter returns the found minimum position P' or, if no improvement is possible, the initial position.

The coordinates of point P are updated by replacing P with P'.

The multidimensional downhill simplex calls **costFunc()**:

```
costFunc (Vertex P):
maxCosts = 0
if search space exceeded return 2 /* 2 is max value
                                    for costs */
for all triangles normals TN adjacent to point P
    if TN not contained in marked bin
        if difference actual TN and original TN >
                                    a return 2
    if TN cotained in marked bin
        if difference TN and original TN > b or
            TN left bin return 2
        else
            c = costs (TN)
            if (c > maxCosts) maxCosts = c
return maxCosts
```

costFunc() checks for violations of the general constraints. If the general constraints regarding search space and maximum tolerated normal changes are violated, maximum costs of 2 are returned to prevent the simplex from further considering the position. Without violations of constraints, costs range from 0 to 1.

a and β are the maximum tolerated normal differences for normals not contained in bins and normals contained in bins, respectively.

costFunc() iteratesover all triangles adjacent to point *P* and calls the costs() function with triangle normals.

Instead of returning maximum cost value, the process could instead sum the costs returned by the costs() call and divide by the number of triangles contributing to costs. This approach, however, showed no gain in performance during testing.

costs (Normal N):

Finally, at the end of the call chain, the costs(Normal N) function calculates costs based on how much normal N contributes to the goal of moving the center of mass in the correct half-plane (half-space in 3D) of the associated bin.

Next, assume that N corresponds to the normal BP_{ij} (the real cost function is called with more parameters identifying the associated bin). Before this calculation N is transformed to the bins' 2D coordinate system, denote the resulting coordinates as $p'_{ij}=(x'_{ij}, y'_{ij})$. *The original* point coordinates before starting the optimization process are $p_{ij} = (x_{ij}, y_{ij})$. Figure 3 illustrates the situation described.

The costs c returned to the caller are calculated as follows:

$$r = \sqrt{x''^2_{ij} + y''^2_{ij}}$$

$$diff = \begin{cases} x_{ij} - x'_{ij} & s_i = 1 \\ x'_{ij} - x_{ij} & s_i = 0 \end{cases} \qquad (1)$$

$$c_1 = |r - r_{init_{ij}}|$$

$$c_2 = \begin{cases} 2 & |diff| > \Delta_{max} \\ \Delta_{max} - diff & 0 \le diff \le \Delta_{max} \\ 1 & -\Delta_{max} \le diff < 0 \end{cases}$$

$$c = w_1 * c_1 + w_2 * c_2$$

with certain weights $w_1, w_2 \in [0,1]$ and $w_1 + w_2 = 1$.

Two components contribute to the costs: the radius with respect to the original point coordinates, and the amount of shift in the x direction, which directly influences the center of mass x coordinate.

A positive amount of shift means the center of mass moves in the direction of the proper half-plane. The more the shift, the less the cost. A push against the proper direction is sanctioned with costs of 1.

Δ_{max} is a constant value that restricts the maximum gain or loss in the x direction. In experiments this value helped to keep points from leaving bins. Exceeding a Δ_{max} gain or loss causes maximum costs of 2.

Without the radius difference being part of the costs function, in tests the algorithm behaved too "greedily" in realizing close to A_{max} gains. It also increased the radius by large amounts and moved the bin's contents closer to the border, where they were likely to leave the bin during mesh altering operations such as polygon simplification.

The actual values of variables used in experiments follow: The optimal values for weights w_1 and w_2, observed in experiments and used in documented test cases, were 0.4 and 0.6, respectively.

The number of iterations (through the point set) in main() was 3, the number of refinements was 2, Δ_{max} was 0.3, a = 5 degrees, β = 10 degrees, and δ remained constant through all iterations. Note that Δ_{max} effectively restricts the amount of normal shift in the x direction for bin *i* by $\Delta_{max} * R_i$, the radius of bin *i*.

After completion of the main function, the model's input mesh had been transformed into an output mesh in which a watermark with contents of bit-string S was embedded.

For watermark retrieval the original center of mass values com_i together with bin center positions BC_i (i = 1, ... N_B) have to be stored, for example in a file.

Retrieving the watermark

Retrieving a watermark requires the reader to know the following a priori information: The number of bins N_B, their positions (bin center normals) BC_i, their radius R_i, and their original center of mass value $com_i = (cx_i, cy_i)$. Also present must be additional information (not specified here) about the enabling precondition for object reorientation. Of course, these values are identical for watermarked copies of one particular model and therefore need only be stored once.

The retrieval algorithm can be described rather briefly, since most of the processing is identical to the embedding process until it reaches the mesh alteration process described in the previous section.

Calculate consistent surface normal patches (R1). Follow the procedure described in stage El.

Transform model into spherical representation (EGI) and adjust orientation (R2). The problems affecting the reorientation process appear in a later section.

Table 1. Experimental results for six test cases.

	Operation	Vertices	Faces	Absolute Loss	Reduction	Relative Loss
Test 1; model 1, bin radius						
20 degrees, 16 of 32 bins	Simplify	2,643	5,282	0	0.67	0.00
	Simplify	1,769	3,534	0	0.45	0.00
	Simplify	1,580	3,156	0	0.40	0.00
	Simplify	1,243	2,482	3	0.32	0.19
	Simplify	700	1,396	4	0.18	0.25
	Crunch	3,326	6,654	0	0.85	0.00
	Crunch	2,730	5,480	2	0.69	0.13
Test 2: model 1, bin radius						
10 degrees, 50 of 100 bins	Simplify	2,913	5,822	0	0.74	0.00
	Simplify	2,641	5,278	1	0.67	0.02
	Simplify	1,451	2,898	4	0.37	0.08
Test 3: model 2, bin radius						
20 degrees, 16 of 32 bins	Simplify	4,718	9,738	0	0.87	0.00
	Simplify	2,795	5,586	0	0.48	0.00
	Simplify	2,289	4,574	1	0.41	0.06
	Crunch	4,661	9,380	0	0.84	0.00
	Crunch	3,598	7,328	1	0.65	0.06
	Crunch	2,799	5,760	4	0.51	0.25
Test 4: model 2, bin radius						
22 degrees, 16 of 16 bins	Simplify	2,048	4,052	0	0.36	0.00
	Crunch	3,153	6,452	0	0.57	0.00
	Crunch	2,473	5,076	1	0.45	0.06
	Crunch	2,166	4,498	3	0.40	0.19
Test 5: model 3, bin radius						
20 degrees, 16 of 32 bins	Simplify	7,395	14,786	0	0.64	0.00
	Simplify	3,029	6,054	3	0.26	0.19
Test 6: model 3, bin radius						
10 degrees, 50 of 100 bins	Simplify	8,353	16,666	0	0.72	0.00
	Simplify	7,467	14.930	3	0.65	0.06
	Simplify	5,991	11,978	7	0.52	0.14
	Simplify	1,743	3,482	12	0.15	0.24
	Simplify	1,617	3,220	16	0.14	0.32
	Crunch	9,770	19,610	2	0.85	0.04
	Crunch	8,072	16,306	6	0.71	0.12

Sample model normals to bins (R3). The model normals are sampled to the bins (a priori data) as described above.

Core watermark retrieval algorithm (R4). The bin contents are transformed from 3D to a 2D representation and the center of mass is calculated as described above.

Denote the calculated center of mass values with $com_i' = (cx_i', cy_i'), i = 1, \ldots . N_B$.

The watermark contents $S' = s_1', \ldots . s_{N_B}', s_i' \in \{0,1\}$, $i = 1, \ldots, N_B$ are simply calculated by

$$s_i' = \begin{cases} 1 & cx_i' > cx_i \\ 0 & cx_i' \le cx_i \end{cases} \qquad (2)$$

In fact, we only test for each bin if a left or right shift of the center of mass with respect to the original value was introduced.

Experimental results

The algorithm just described has been implemented in C + + using the graphics package MAM/VRS (Modeling and Animation Machine/Virtual Renderings System).[11] The downhill simplex implementation came from the second edition of *Numerical Recipes in C—The Art of Scientifc Programming*[10] with only slight modifications.

To test the algorithm's properties for robustness against polygon simplification, I applied two simplification programs, plycrunch and plysimplify, to the watermarked model of Wagner's bust. Both programs belong to the software distribution of the Simplification Envelopes implementation.[12,13]

- plycrunch simplifies an object by sharing nearby vertices. This method does not preserve the topology of the original model.
- plysimplify attempts to minimize the total number of polygons while guaranteeing that all points of an approximation lie within a user-specifiable distance from the original model and vice versa. plysimplify preserves the topology.

The three models of Wagner's bust used for the test series were all polygon reduced versions of the laser-scanned original:

- Model 1: 3,923 vertices, 7,842 faces (triangles)
- Model 2: 5,616 vertices, 11,228 faces
- Model 3: 11,488 vertices, 22,972 faces

Table 1 summarizes the results of six test cases. It lists the operation applied, number of vertices and faces after applying the operation, absolute loss of watermark bits, relative loss, and reduction, which is simply the quotient of the number of faces after and before applying the operation. The model simplified precedes each list of results, along with the radius common to all bins. "*m* of *n*" bins means that the unit sphere was tesselated with approximations of *n* evenly distributed bin centers from which *m* were randomly chosen for embedding watermark bits. The bins did not overlap. The bin centers were calculated with the application tessel,[14] which tessellates a sphere with points while trying to minimize maximum distances of each pair of points. For testing, tessellations with 16, 32, and 100 points (normals) were generated. From these 16 (tests 1, 3, 4, 5) or 50 (tests 2, 6) bins were selected randomly for embedding watermark bits. In no test case did the bins cover all of the models' surface normals.

Figures 4, 5, and 6 show the models for test case 4. This test case produced the highest observed level of robustness. However, as you can see in the left image of Figure 5, a trade-off of visual quality resulted. The regions around the mouth were distorted, for example. One reason this trade-off occurred was because of the maximum bin size, with a 22-degree radius. Thus, in this test, a maximum amount of the object's surface was contained in bins and subject to alteration by the embedding process.

The other reason for the trade-off was driven by the fact that, if the number of elements in a bin increases, the number of conflicts during optimization increases. Conflicts arise when constraints regarding maximum tolerated angle differences prevent normals of marked bins from being pushed in their proper directions or when the several push directions show up as contradictory.

For a smaller bin radius, visual quality improves, but robustness decreases. You can see this in Figure 7 and for test case 6 in Table 1.

Note that in all tests performed, the watermark bits remained after the embedding process. This result was not necessarily expected because, as already stated, the watermark embedding algorithm bears certain conflicts. To guarantee this stability, it sufficed to select bins containing a number of normals large enough to compensate for normals entering during the embedding process.

Coding a 0 or 1 in a bin presents two different optimization problems. In general, embedding different bit strings presents different optimization problems. Nevertheless, the tests showed no evidence that either of the two values was more stable or that one bit string remained less stable than the others. This property is important for embedding arbitrary bit strings at constant positions.

I must stress that, when selecting the embedding positions, I did not take into account properties of the reduction methods applied. The results should reveal if the embedding algorithm holds general potential against mesh simplification methods. The crunch results, for example, improve significantly if bins are selected whose contents are mostly located in low curvature regions. For the test model these triangles, were larger in size, causing larger vertex distances. This property leaves a higher percentage of the bins' vertices untouched by the crunch algorithm.

4 The original bust with 11,488 vertices and 22,972 faces (test case 4).

5 The left image shows the bust after embedding a watermark of 16 bits in length. The bin radius was 22 degrees, and 16 of 16 equally spaced bins were selected. The right image shows triangles contained in selected bins (red) and triangles whose normal changed by more than 0.5 degrees and not contained in selected bins (green).

6 The left image shows the watermarked bust after applying plycrunch (size was reduced to 3,153 vertices and 6,452 triangles). The right image shows the bust after applying plysimplify (2,048 vertices, 4,052 triangles). In both cases the watermark was still present (test case 4).

7 The bust after embedding a 50-bit watermark (test case 6). The bin size was 10 degrees, and 50 of 100 equally spaced bins were selected. Red faces represent triangles contained in the selected bins; green faces represent triangles whose normal changed by more than 0.5 degree and not contained in a selected bin.

Some bins showed more robustness than others. Their bin value remained stable up to a certain simplification ate. Interestingly, for the majority of bins the (false) alue did not change again at higher simplification rates.

The results could have improved by repeating the mbedding process with the bins that showed themelves to be stable after mesh alteration. The problem—and subject of further research-is to identify the stable bins prior to mesh alteration.

Reorienting watermarked models

The watermark retrieval process is preceded by an bject recognition and orientation stage (not the subect of this article). To discover the demands on the precision of the orientation process, I reoriented the watermarked object before performing the extraction rocess. The required precision for retrieving the watermark without loss of information varied between 2 and 3 degrees, depending on bin radius.

I am developing an orientation module that tries to find a complete subgraph, consisting of three to five vertices, of the original model EGI in the watermarked node1 EGI representation. Each vertex of this graph corresponds to a bin center, and each edge length corresponds to angle differences of centers in the EGI representation. After matching the subgraph, the module must find the correct orientation by testing orientations in a small range (targeted 10 degrees). The decision as to when orientation is correct or sufficient can be based on detecting, for example, a constant portion of the embedded watermark.

If the original is accessible during the orientation process, several subgraphs could be generated on the fly from the original model and matched against the watermarked copy. This aspect proves important in cases where the watermarked copy consists only of parts of the original model. The regions important to precomputed subgraphs might be missing in this case.

Compensating for failure rates

If you use the algorithm for embedding arbitrary information (bit strings), techniques to compensate for failure rates include

- Increase information redundancy. Repeat the bit string *n* times and do "majority voting" in the retrieval process.
- Apply error-correcting codes.

The need for embedding arbitrary information arises mainly for two cases: embedding information allowing proof of ownership (a hash of the original model) and public watermarks (for example, a URL or IP address of the creator/copyright holder's Web site).

If the watermarkjust needs to point to external information, as when embedding tracing information such as the model licensee's identity, you can apply the following techniques:

- Instead of embedding things like consecutive serial numbers, embed random bit strings. Map these strings to such things as actual serial numbers or customer information. In the retrieval process, decode bit string to string with the minimum hamming distance.
- Seed a random number generator with information. The output determines embedding positions (bin centers) or, for example, the rotation angle of a predefined set of positions. Embed a "010101" sequence, for example. During the retrieval process, check for all possible watermarks. Select the one with the minimum hamming distance to sequence.

Improving the watermarking algorithm

The watermarking algorithm as described and implemented can be considered a bare bones solution. A variety of possibilities exist for improving it:

- Each normal contained in a selected (meaning used for embedding) bin could be assigned a weight corresponding to its surface relative to the total surface of the bin to which it belongs. This weight should be used in cost function and center of mass calculations. Currently, all normals in a bin contribute equally to the center of mass calculation. Assigning a weight to each normal would let the algorithm modify far fewer normals to achieve the desired center of mass deviation. It would sufficice, for example, to change the normals with the largest weights that occupied 30 percent of the bin's total surface. The disadvantage is the increased sensitivity of the bins center of mass with respect to normals with large weights entering or leaving the bin.
- Select bins whose contents are (mostly) located in regions of the model that can be considered perceptually important features. A good starting point would be to select regions containing or located near sharp edges with reasonably large surface sizes.
- Take the surroundings into consideration when choosing a bin center position and radius. If the majority of normals occupy the outermost regions of the bin and a large number of normals lie outside, at the bin's border, the center of mass will change significantly due to normal exchanges with the bins surroundings. If weights are assigned to normals, the bin center must be located in a position such that normals with the largest weights lie near the center and others outside the bin lie far from the border.
- Some bins showed outstanding robustness in tests compared to others. Reverse engineering might lead to reliable criteria for ad hoc judging of a bins quality.
- Reduce the contribution of normals lying far from the bin center in the center of mass calculation. Normals with large distances will likely leave the bin in response to mesh manipulations. Alternatively, points not contained in a bin but near the bin's border are likely to enter the bin. A simple step along these lines during testing did not improve the results: Before calculating the center of mass, all points with a radius greater than R_1 ($R_1 < 1$) were dragged (orthogonal projected) onto the border circle with radius R_1 to reduce their x- and y-values and thus their contribution to the center of mass.
- As ⊠□◆ can ▪ℳℳ in Figure 1, the regions for coding 0 and 1 bits differ in size. Although not observed in tests, this

might result in a coded 0 bit being more stable than a coded 1 bit for the same bin or vice versa (assuming modification processes introduce random changes in center of mass values). The regions can be adjusted to equal size simply by rotating the bin contents around the center by an amount that places the center of mass on the x-axis.

Conclusion

I proposed a watermarking algorithm to embed private watermarks. The algorithm showed promising potential with respect to robustness against mesh simplifications.

The highest rate of robustness achieved was resistance to a simplification that reduced the model to 36 percent of its original number of faces (test case 4). As already mentioned, many possibilities exist for improvements and further fine-tuning. Since bins were selected randomly as embedding positions in test cases, a more elaborate strategy for selection of embedding positions should improve the results.

One drawback of the algorithm is the large amount of a priori data needed before watermark retrieval. For private watermarks, this is tolerable. A more relevant drawback is the amount of preprocessing needed before the watermarking core algorithm can be applied.

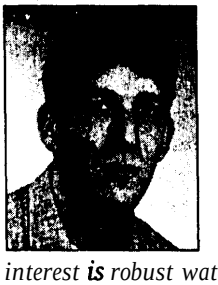
This preprocessing involves reorientation. The demands on this process regarding accuracy are quite relaxed. A first broad orientation with an error of 10 degrees with respect to the original orientation should suffice. In a second orientation stage the precise orientation (with two to three degrees of error in documented test cases) would be determined by random testing. To recognize the correct orientation, a constant sequence could be embedded in a portion of the watermark. The number of correct bits with respect to this sequence increases near the correct orientation.

The watermarking system proposed doesn't suit embedding public watermarks because the system cannot be realized without a certain amount of a priori data that must be known to the reader in advance of watermark retrieval. We may use fixed values for the positions of bin centers and radius, and even for orientation (always orient the model with respect to directions from the center of mass-vertices-to the furthermost and nearest vertex), but the original center of mass values are still required as reference values for watermark retrieval. One possibility, of course, would be to tessellate bins in the 0 and 1 regions. The optimization problem then changes from pushing the center of mass in a certain half-plane (2D) to moving the center of mass in the region that encodes the desired bit value. Even if this works, the grade of robustness would be minimal.

This article should increase the focus on surface normal distribution as a feature suitable for embedding private watermarks with the potential for resistance against various polygon altering operations. In the case of polygon simplification, the tests performed have proven the algorithm's potential. Future work will include improving the watermarking system's robustness against cut operations and more severe attacks such as nonuniform scaling. ■

References

1. E. Koch and J. Zhao, "Towards Robust and Hidden Image Copyright Labeling," *Proc.* 1995 *IEEE Workshop on Nonlinear* Signal *and Image Processing,* June 1995, IEEE CS Press, Los Alamitos, Calif., pp. 452-455.
2. J. Zhao and E. Koch, "A Digital Watermarking System for Multimedia Copyright Protection," *Proc. ACM Multimedia* 96, ACM Press, New York, Nov. 1996, pp. 443-444.
3. A.G. Bors and I. Pitas, "Image Watermarking using DCT Domain Constraints," *IEEE Int. Conf.* on Image *Processing,* IEEE Press, Piscataway, N.J., 1996, pp. 231-234.
4. I. Cox et al., "Secure Spread Spectrum Watermarking for Images, Audio, and Video," *IEEE Int'l Conf.* Image Processing, Vol. 3, IEEE Press, Piscataway, N.J., 1996, pp. 243-246.
5. B. Hartung, F. Hartung, and B. Girod, "Digital Watermarking of Raw and Compressed Video," *Proc. European EOS/SPIE S'p. Advanced Imaging and Network Technologies,* Society of Photo-Optical Instrumentation Engineers, Bellingham, Wash., 1996.
6. L. Boney, A.I. Tewfik, and K.I. Hamdy, "Digital Watermarks for Audio Signals," *Proc. EUSIPCO* 96, Vol. III, VIII European Signal Processing Conf., 1996, pp. 1697-1700.
7. R. Ohbuchi, H. Masuda, and N. Aono, "Watermarking Three-Dimensional Polygonal Models," *ACM Multimedia 97,* 1997, ACM Press, New York, pp. 261-272.
8. B.K.P. Horn, *Robot Vision, The* MIT Electrical Engineering and Computing Series, MIT Press, Cambridge, Mass., 1986.
9. K. Ikeuchi, *Determining Attitude of Object from Needle Map Using Extended Gaussian Image,* MIT, A.I. Memo No. 714, Cambridge, Mass., Apr. 1983.
10. W.H. Press et *al., Numerical Recipes in C—The Art of Scientific Programming,* 2nd Ed., "Section 10.4: Downhill Simplex Method in Multidimensions," Cambridge University Press, Cambridge, UK, 1992, pp. 408-412.
11. J. Doellner and K. Hinrichs, MAM/VRS—Modeling and Animation Machine/Virtual Renderings Sys. V. 2.0, http://wwwmath.uni-muenster.de/math/inst/info/u/mam.
12. J. Cohen et al., "Simplification Envelopes," *Computer Graphics Proc.,* Ann. Conf. Series, ACM Siggraph, ACM Press, New York, 1996, pp. 119-128.
13. Simplification Envelopes V. 1.2 implementation, http://www.cs.unc.edu/~geom/envelope.html.
14. Tessel application V. 2.0, http://www3.uniovi.es/~quimica.fisica/qcg/tessel/tessel.html.



Oliver Benedens works as a *researcher at the Fraunhofer Institute* for Computer Graphics *in Darmstadt, Germany. He received a university degree* in computer science *at the Technical University of Darmstadt in 1996. His main research interest is robust watermarking techniques focusing on 3D data. Other interests coversecure communications and the field ofelectronic commerce.*

Readers may contact Benedens at Fraunhofer Institute of *Computer Graphics, Rundeturmstrasse 6, 064283 Darmstadt, Germany, e-mail* benedens@igd.fhg.de.